#3
3-9-01

Docket No.: 121.1010/HEW

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent Application of:)
)
NISHIGAYA et al.)
) Group Art Unit: To be assigned
Serial No.: To be assigned)
)
) Examiner: To be assigned
Filed: Herewith)

For: METHOD AND APPARATUS FOR DETERMINING DYNAMIC FLOW AND COMPUTER-READABLE MEDIUM STORING A PROGRAM FOR PERFORMING THE METHOD

SUBMISSION OF CERTIFIED COPY OF PRIOR FOREIGN APPLICATION IN ACCORDANCE WITH THE REQUIREMENTS OF 37 C.F.R. §1.55

Assistant Commissioner for Patents
Washington, D.C. 2023l

Sir:

In accordance with the provisions of 37 C.F.R. §1.55, the Applicants submit herewith a certified copy of the following foreign application:

Japanese Patent Application No. 2000-015386
Filed: January 25, 2000

It is respectfully requested that the applicant(s) be given the benefit of the foreign filing date as evidenced by the certified papers attached hereto, in accordance with the requirements of 35 U.S.C. §119.

Respectfully submitted,

STAAS & HALSEY LLP

Date: January 2, 2001 By: Heath E. Wells

Heath E. Wells
Registration No. 43,257

700 11th Street, N.W., Ste. 500
Washington, D.C. 20001
(202) 434-1500

日 本 国 特 許 庁
PATENT OFFICE
JAPANESE GOVERNMENT

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office.

出 願 年 月 日
Date of Application: ２０００年 １月２５日

出 願 番 号
Application Number: 特願２０００－０１５３８６

出 願 人
Applicant (s): 富士通株式会社

２０００年 ５月１２日

特 許 庁 長 官
Commissioner,
Patent Office 近 藤 隆 彦

出証番号 出証特２０００－３０３２２９０

【書類名】　特許願

【整理番号】　9951517

【提出日】　平成12年 1月25日

【あて先】　特許庁長官殿

【国際特許分類】　G06F　9/44

【発明の名称】　アクション連鎖による動的データフロー決定装置，方法およびコンピュータ読み取り可能な記録媒体

【請求項の数】　9

【発明者】

【住所又は居所】　神奈川県川崎市中原区上小田中４丁目１番１号　富士通株式会社内

【氏名】　西ヶ谷　岳

【発明者】

【住所又は居所】　神奈川県川崎市中原区上小田中４丁目１番１号　富士通株式会社内

【氏名】　福田　茂紀

【特許出願人】

【識別番号】　000005223

【氏名又は名称】　富士通株式会社

【代理人】

【識別番号】　100087848

【弁理士】

【氏名又は名称】　小笠原　吉義

【電話番号】　03-3807-1151

【選任した代理人】

【識別番号】　100074848

【弁理士】

【氏名又は名称】　森田　寛

【選任した代理人】

【識別番号】　100087147

【弁理士】

【氏名又は名称】　長谷川　文廣

【手数料の表示】

【予納台帳番号】　012586

【納付金額】　21,000円

【提出物件の目録】

【物件名】　明細書　1

【物件名】　図面　1

【物件名】　要約書　1

【包括委任状番号】　9707817

【プルーフの要否】　要

【書類名】　明細書

【発明の名称】　アクション連鎖による動的データフロー決定装置，方法およびコンピュータ読み取り可能な記録媒体

【特許請求の範囲】

【請求項１】　分散システムにおけるイベント処理においてアクション連鎖により動的にデータフローを決定する装置であって，

イベントオブジェクト受信時に実行されるアクションに関する情報を，サーバオブジェクトと分離して記憶するアクション・属性記憶部と，

受信したイベントオブジェクトのタイプにより，発火すべきアクションを前記アクション・属性記憶部に記憶されるアクションの中から選択するフロー制御部とを備え，

前記フロー制御部によってアクション連鎖を実現する

ことを特徴とするアクション連鎖による動的データフロー決定装置。

【請求項２】　請求項１に記載のアクション連鎖による動的データフロー決定装置において，

前記アクション・属性記憶部は，イベントオブジェクト受信時に実行されるアクションの定義とそのアクションが選択される条件としての入力パターンの定義とを分離した形で記憶し，

入力パターンの定義の変更によって，アクションの定義または構成が変更されなくてもイベントに対する振る舞いが変更されるようにした

ことを特徴とするアクション連鎖による動的データフロー決定装置。

【請求項３】　請求項１に記載のアクション連鎖による動的データフロー決定装置において，

前記フロー制御部は，イベント受信時に実行されたアクションがその実行結果としてイベントをオブジェクトを返した場合，新たに受信したイベントオブジェクトのタイプをチェックし，次に発火すべきアクションの選択，実行を繰り返すことにより，動的なデータフローを決定する

ことを特徴とするアクション連鎖による動的データフロー決定装置。

【請求項４】　請求項２に記載のアクション連鎖による動的データフロー決

定装置において，

前記アクション・属性記憶部に記憶される入力パターンの定義に，イベントオブジェクトのタイプだけでなく，イベントオブジェクトの値またはイベントオブジェクトが持つ属性の値を設定可能に構成され，これらの入力パターンの定義によってアクションの発火が制御される

ことを特徴とするアクション連鎖による動的データフロー決定装置。

【請求項５】　請求項２に記載のアクション連鎖による動的データフロー決定装置において，

前記アクション・属性記憶部に記憶される入力パターンの定義に，直前に実行されることが期待されるアクションの名前を設定可能に構成され，

前記フロー制御手段は，アクションの選択時に前記入力パターンの定義をチェックすることにより，実行されるアクションの順序を制御する

ことを特徴とするアクション連鎖による動的データフロー決定装置。

【請求項６】　請求項３に記載のアクション連鎖による動的データフロー決定装置において，

前記フロー制御手段は，アクションの選択を行う際に既に実行したアクションのリストを記憶し，一度実行したアクションは発火の対象から外すことにより，データフローの無限ループを避けるようにした

ことを特徴とするアクション連鎖による動的データフロー決定装置。

【請求項７】　分散システムにおけるイベント処理においてアクション連鎖により動的にデータフローを決定する装置であって，

アクションに関する定義情報を記憶するアクション・属性記憶部と，

メッセージを受信するメッセージ受信部と，

メッセージを送信するメッセージ送信部と，

受信したメッセージが前記アクションに関する定義情報を変更する要求であった場合に，そのアクションに関する定義情報を変更するアクション管理部と，

アクション実行要求として受信されたメッセージのパラメータ並びと，前記アクション・属性記憶部に記憶された情報との照合により，マッチするアクションを選択するパターンマッチ処理部と，

選択されたアクションの実行を管理するアクション実行部と，

イベントオブジェクト受信時に前記アクション実行部によって起動され，受信したイベントオブジェクトのタイプにより次に発火すべきアクションの選択，実行の制御を行うフロー制御部とを備える

ことを特徴とするアクション連鎖による動的データフロー決定装置。

【請求項８】　分散システムにおけるイベント処理においてアクション連鎖により動的にデータフローを決定する方法であって，

イベントオブジェクト受信時に実行されるアクションとサーバオブジェクトとを分離し，

受信したイベントオブジェクトのタイプにより発火すべきアクションを選択するフロー制御手段をサーバオブジェクトに設け，このフロー制御手段によって，次に発火すべきアクションの選択，実行の制御を行い，

サーバオブジェクトにおける他のアクションに影響を与えることなく，動的に新たなイベント処理を導入する

ことを特徴とするアクション連鎖による動的データフロー決定方法。

【請求項９】　分散システムにおけるイベント処理においてアクション連鎖により動的にデータフローを決定する装置をコンピュータによって実現するためのプログラムを記録した記録媒体であって，

メッセージを受信するメッセージ受信処理と，

メッセージを送信するメッセージ送信処理と，

受信したメッセージがアクションに関する定義情報を変更する要求であった場合に，そのアクションに関する定義情報を変更するアクション管理処理と，

アクション実行要求として受信されたメッセージのパラメータ並びと，アクションを選択するためのパターンとの照合により，マッチするアクションを選択するパターンマッチ処理と，

選択されたアクションの実行を管理するアクション実行処理と，

イベントオブジェクト受信時によって起動され，受信したイベントオブジェクトのタイプにより次に発火すべきアクションの選択，実行の制御を行うフロー制御の処理とを，

コンピュータに実行させるプログラムを記録した

ことを特徴とするコンピュータ読み取り可能な記録媒体。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】

本発明は，ネットワークで接続された複数のコンピュータが連携する分散型システムにおけるイベント処理に係り，特に柔軟なアクションの連鎖を可能にしたアクション連鎖による動的データフロー決定装置に関する。適用可能な技術としては，CORBA ，DCOM，Java RMIなどの分散オブジェクト指向プラットフォームがある。

【０００２】

【従来の技術】

分散処理システムの開発において，イベント処理モデルは重要な役割を持っている。適切なルールに基づいたイベント処理モデルを用いて，互いに連携するオブジェクトのインタフェースを設計することにより，大規模で複雑な分散システムになった場合でも，局所的な振る舞いを把握することが可能となる。これにより，新たなオブジェクトをシステムに追加して，機能を拡張したり，不要な部分を切り離すなどの手続きを容易にすることができる。

【０００３】

従来の分散システムにおける一般的なイベント処理モデルについて説明する。イベントを配送するオブジェクトはイベントソースと呼ばれ，イベントを受信して何らかの処理を行うオブジェクトはイベントリスナと呼ばれる。

【０００４】

図１５に，従来のイベント処理モデルに基づくクラス定義の例を示し，図１６に，そのイベント処理シーケンスの例を示す。イベントソースとなるオブジェクト(Source object) は，少なくともイベントリスナを外部から登録するためのインタフェース（ここでは，addEventListener()）を持ち，イベントリスナとなるオブジェクト（Listener object ）は，イベントが発生したときに実行されるアクション・メソッド（ここでは，action()）を持つ。

【０００５】

イベントリスナの参照ポインタがaddEventListener()によってイベントソース（Source object ）に登録されると，イベントソースは，何らかの状態変化が起こったとき（status changed）に，登録されたイベントリスナの参照ポインタを使って，予め決められたアクション・メソッドの呼び出しを行う。これにより，イベントの発生に連動した処理を分散システム上で行うことができる。

【０００６】

このモデルの問題点は，イベントが発生したときに実行される処理がイベントリスナのメソッドとして定義されているため，実行中にイベントリスナの振る舞いを変更することができないということである。もう一つの問題点は，それぞれのオブジェクトがイベントソースとして振る舞うか，イベントリスナとして振る舞うかは，それぞれのオブジェクトの設計時に決定され，実行時にはその役割を変更できないことである。

【０００７】

【発明が解決しようとする課題】

従来のイベント処理モデルの問題点を解決するために考案されたのが，エージェント・アクション・モデル（アクション分離型イベント処理モデル）である。図１７に，アクション分離型イベント処理モデルにもとづくクラス定義の例を示し，図１８に，そのイベント処理シーケンスの例を示す。

【０００８】

このモデルは，イベントリスナと動作の定義であるアクションとをオブジェクトのレベルで分離し，イベントが発生したときの振る舞いを実行時に変えることができるようにしたものである。イベントソースおよびイベントリスナは，ソース・エージェント（Source agent）およびリスナ・エージェント（Listener agent）という汎用の雛型によって実現される。

【０００９】

前記のイベント処理モデルでは，イベントソースは，予め決められたイベントリスナのメソッドを呼ぶことによってイベント処理が行われていたが，エージェント・アクション・モデルでは，特定のメソッドを呼ぶのではなく，イベントリ

スナとなるエージェント（リスナ・エージェント）に対してイベントだけをメッセージとして送信する。リスナ・エージェントは受信したメッセージにマッチするアクションを選択，実行するため，アクションの入力パターンが同じで，実装が異なるアクションを置き換えることにより，イベント受信時の振る舞いを変更することが可能となる。また，イベントソースとなるオブジェクトもイベントリスナとなるオブジェクトも同じエージェントから派生するため，オブジェクトレベルでの区別はなく，アクションの組み合わせによって，動的にその役割を決定することができる。

【００１０】

しかし，エージェント・アクション・モデルにも欠点がある。それは，イベントの発生によって実行される処理を，図１９に示すような複数のアクションの連鎖で表現することが難しいという点である。このようなアクションの連鎖によるデータフローは，例えば図２０に示すように，アクション“Ａ＃１”の内部で次のアクション“Ａ＃４”を呼ぶように記述（call("A#4");）することで実現することは可能である。しかし，特定の状況でのみ有効なコードがアクションに埋め込まれるため，各アクションの汎用性や独立性が損なわれてしまい好ましくない。

【００１１】

また，アクションの呼び出しは，ＲＰＣ(Remote Procedure Call）やメソッド呼び出しと同じように同期的であるため，ｎ個のアクションの連鎖が実行されるときには，ｎ個のスレッドが生成されることになり，アクションの連鎖が長くなるほど，メモリやＣＰＵにかかる負荷を大きくしてしまうことも問題となる。

【００１２】

【課題を解決するための手段】

上記の問題点を解決するため，本発明は，イベントの発生に同期して実行される処理を，エージェント・アクション・モデルにおけるアクションの連鎖によるデータフローモデルで実現し，従来のエージェントの構成において，受信したイベントオブジェクトのタイプにより発火すべきアクションを選択し，データフローの実行管理を行うフロー制御手段を設けることを特徴とする。

【００１３】

このフロー制御手段は，イベントの種類や状態に応じて，アクション・属性記憶領域内に存在するアクションの選択・実行を繰り返すことによりアクション連鎖を生み出し，動的なデータフローを決定する。

【００１４】

このフロー制御手段は，アクション・属性記憶領域内に特別なアクションの一つとして実装することができる。

【００１５】

本発明は，以下のように作用する。他のエージェントからのメッセージがアクション実行要求の場合には，アクション実行手段へメッセージが転送される。アクション実行手段では，受信したメッセージの実行要求のパラメータに指定された名前のアクションが，アクション・属性記憶領域内に存在するかどうかを調べ，同名のアクションが存在すれば，そのアクションを実行し，処理が完了するのを待つ。アクションの実行が完了したら，実行結果を要求元エージェントに返送する。

【００１６】

ここで，アクション実行手段は，実行要求のパラメータの指定が，イベントオブジェクトである場合に，フロー制御手段を起動する。

【００１７】

フロー制御手段では，実行要求のパラメータであるイベントオブジェクトをもとにメッセージパターンを作成し，送信先アドレスを自分自身に設定したアクション実行要求をメッセージ送信手段に送信して実行結果を待つ。実行結果はアクション実行手段を介して返送される。以降も同様の処理を繰り返す。

【００１８】

すなわち，イベントオブジェクトをもとにメッセージパターンを作成し，アクション実行要求を発行する。もし，返送され実行結果が，アクション名のリストであった場合には，リストされているアクションを並列に実行するため，イベントオブジェクトを複製し，それぞれのアクション名をパラメータとしてフロー制御手段の実行要求を再帰的に行う。返送された実行結果になんのデータも含まれ

ていなければ，処理を終了してアクションの連鎖実行を停止する。

【００１９】

このように，本発明では，データフローを構成するアクションの独立性を確保し，汎用に作られたアクション部品を組み合わせることで，容易にイベント処理手続きを定義し，また，いったん構築したイベント処理手続きのカスタマイズも容易に行えるようにする。

【００２０】

また，アクション連鎖の決定には，アクションが扱うことができるイベントオブジェクトのタイプを示した入力パターンと実際に発生したイベントオブジェクトのタイプの整合性をチェックすることで，ある程度，システムが適切なフローを決定できるため，データフローの設計者の負担を軽減し，誤ったフローを定義してしまうことを避ける効果がある。また，アクションの定義とその入力パターンの定義とを分離することにより，広い範囲で適用可能なアクションの実装を特定のイベントタイプのためのフローに適用するには，そのアクションの入力パターンのイベントタイプを狭いタイプに設定するだけでよく，そのアクションの実装を変更する必要がない。

【００２１】

また，入力パターンの定義として，イベントのタイプだけではなく，受信したイベントオブジェクトが持つ属性の値や，直前に実行されたアクションの名前やタイプなどを必要に応じて定義できるようにすることにより，データフローにおけるアクションの順序を制御することができる。

【００２２】

【発明の実施の形態】

以下，本発明の実施の形態を説明する。

【００２３】

図１は，本発明システム構成例を示す図である。本発明に係る動的データフロー決定装置（エージェント）１は，メッセージ受信部１１，アクション管理部１２，パターンマッチ処理部１３，アクション実行部１４，アクション・属性記憶部１５，メッセージ送信部１６を備え，アクション・属性記憶部１５には，各ア

クション（アクション＃１～＃ｎ）の他に，本発明の特徴であるフロー制御部２０を備える。

【００２４】

メッセージ受信部１１は，他のエージェントから送信されたメッセージを一時的に蓄えるキューを持ち，メッセージの種類によって，エージェント内部の個別の処理部（モジュール）に振り分ける。アクション管理部１２は，エージェントが持つアクションの構成を変更する要求を外部から受け付け，要求に基づいてアクション構成を変更する。パターンマッチ処理部１３は，アクション実行要求として受信されたメッセージのパラメータ並びとエージェントに保存されているアクションの入力パターンを比較し，マッチするアクションを選択する。アクション実行部１４は，アクション・属性記憶部１５に保存されたアクションの実行を管理する。アクション・属性記憶部１５は，各アクションおよびフロー制御部（アクション）２０のコードを記憶する領域である。メッセージ送信部１６は，アクション管理に関するメッセージ，実行要求などを受信し，該当するエージェントに要求されたメッセージを送信する。

【００２５】

フロー制御部２０は，例えばアクションの一つとして実現され，アクション実行部１４から実行要求のパラメータであるイベントオブジェクトを受けて，自分自身を送信先アドレスに設定したアクション実行要求を作成してメッセージ送信部１６に送信し，実行結果を待ち，実行結果である新たなイベントオブジェクトを受けて再度実行要求を作成し送信するという処理を繰り返すことによりフローを制御する。すなわち，フロー制御部２０は，受信したメッセージがイベントオブジェクトであった場合に起動され，イベントの種類や状態に応じて，エージェント内に存在するアクション部品の選択，実行を繰り返すことにより，アクション連鎖を生み出し，動的なデータフローを決定する機能を持つ。

【００２６】

なお，図２に示すように，アクション・属性記憶部１５に，変更可能な入力パターン２１’を，アクション（＃１～＃ｎ）毎の記憶領域内にアクションとは別に設けてもよい。これは，アクションの実装が，コンパイル済みのプログラムで

ある場合であっても，そのアクションを入れ替えることなく，入力パターンを変更できるようにする場合に有効である。

【００２７】

また，図３に示すように，アクションごとにパターンマッチルール１３２を記憶する領域をアクション・属性記憶部１５に設けてもよい。この場合，パターンマッチ処理部１３は，各アクションが保持するパターンマッチルール１３２を参照して，入力パターンの適応性をチェックする。こうすることにより，アクションの置き換えの手軽さで，パターンマッチのルールを拡張することが可能となる。

【００２８】

以下に，各手段の処理の流れを説明する。図４に，メッセージ受信部１１の処理フローを示す。メッセージ受信部１１では，メッセージの種別を判断し（ステップＳ１），受信したメッセージがアクションの追加／削除／置換要求だった場合には，そのメッセージをアクション管理部１２に転送する（ステップＳ２）。受信したメッセージがアクション実行結果であった場合には，アクション実行部１４へ転送する（ステップＳ３）。受信したメッセージがアクション実行要求であった場合には，そのメッセージをパターンマッチ処理部１３に転送する（ステップＳ４）。

【００２９】

図５に，アクション管理部１２の処理フローを示す。アクション管理部１２では，受信待ち状態において，受信したメッセージの処理種別を判断し（ステップＳ２１），アクション追加要求（パラメータ：名前，アクション）を受け取った場合には，追加要求のパラメータである名前（アクション名）と同じ名前のアクションがアクションリスト保存領域に存在するかどうかをチェックし（ステップＳ２２），もし，同名のアクションが存在するならエラーメッセージをメッセージ送信部１６に転送して要求元エージェントに返送する（ステップＳ２３）。そうでなければ，追加要求のパラメータであるアクションを指定された名前でアクション・属性記憶部１５に追加保存する（ステップＳ２４）。

【００３０】

アクション管理部１２がアクション置換要求（パラメータ：名前，アクション）を受け取った場合には，置換要求のパラメータである名前（アクション名）と同じ名前のアクションがアクションリスト保存領域に存在するかどうかをチェックし（ステップＳ２５），もし，同名のアクションが存在しないならエラーメッセージを要求元エージェントに返送する（ステップＳ２３）。そうでなければ，同名のアクションをアクション・属性記憶部１５から削除し（ステップＳ２６），置換要求のパラメータであるアクションを同名のアクションとしてアクション・属性記憶部１５に追加保存する（ステップＳ２７）。

【００３１】

アクション管理部１２がアクション削除要求（パラメータ：名前）を受け取った場合には，削除要求のパラメータである名前（アクション名）と同じ名前のアクションがアクション・属性記憶部１５に存在するかどうかをチェックし（ステップＳ２８），もし，同名のアクションが存在しないならエラーメッセージを要求元エージェントに返送する（ステップＳ２３）。そうでなければ，同名のアクションをアクション・属性記憶部１５から削除する（ステップＳ２９）。

【００３２】

図６に，パターンマッチ処理部１３の処理フローを示す。パターンマッチ処理部１３では，アクション実行要求（パラメータ）を受信すると，アクションリストを空にし（ステップＳ３０），全てのアクションに対して，受信されたメッセージのパラメータ並びと各アクションの入力パターンの整合性をチェックし（ステップＳ３１），適用可能なアクションの名前をアクションリストに追加する（ステップＳ３２）。全てのアクションをチェック後，一つもアクションが選択されなかった場合には，エラーメッセージを送信元エージェントに返送する（ステップＳ３３）。また，選択されたアクションが一つだけであった場合には，そのアクションの実行要求をアクション実行部１４に送信する（ステップＳ３４）。また，複数のアクションが選択された場合には，そのアクションリストを送信元エージェントに返送する（ステップＳ３５）。

【００３３】

図７に，アクション実行部１４の処理フローを示す。アクション実行部１４で

は，実行要求を受信すると，実行要求のパラメータに指定された名前のアクションがアクション・属性記憶部１５に存在するか否かをチェックし（ステップＳ４０），もし存在しないならエラーメッセージを要求元エージェントに返送する（ステップＳ４１）。もし，同名のアクションが存在すれば，そのアクションを実行し（ステップＳ４２），処理が完了するのを待つ（ステップＳ４３）。アクションの実行が完了したら，実行結果を要求元エージェントに返送する（ステップＳ４４）。また，アクション実行部１４が，アクションの実行結果を受信すると，受信した実行結果を待っている要求元のアクションに実行結果を通知する（ステップＳ４５）。

【００３４】

ここで，実行要求のメッセージのパラメータがイベントオブジェクトの場合には，発火するアクションとして，フロー制御部２０が起動され，動的なフロー制御が行われることになる。

【００３５】

図８に，フロー制御部２０の処理フローを示す。アクション実行部１４により，フロー制御部２０が起動されると，フロー制御部２０は，実行要求のパラメータであるイベントオブジェクトをもとにメッセージパターンを作成し（ステップＳ５０），送信先アドレスを自分自身に設定したアクション実行要求をメッセージ送信部１６に送信して（ステップＳ５１），実行の処理結果を待つ（ステップＳ５２）。その処理結果は，アクション実行部１４によりフロー制御部２０に返送され，再びフロー制御機能の処理が継続される。

【００３６】

フロー制御部２０では，処理結果があれば（ステップＳ５３），処理結果の種類を検査し（ステップＳ５４），もし，それがイベントオブジェクトであった場合には，上記のステップＳ５０～Ｓ５４の処理を繰り返す。すなわち，イベントオブジェクトをもとにメッセージパターンを作成し，アクション実行要求を発行する。ここで，不要なアクションの発火を避けるため，イベントオブジェクトに実行済みアクションリストを保存するメモリを設け，直前に実行したアクションの名前を実行済みアクションリストに保存するようにしてもよい（ステップＳ５

５）。

【００３７】

もし，返送された実行結果がアクション名のリストであった場合には，リストされているアクションを並列に実行するため，イベントオブジェクトを複製し（ステップＳ５７），それぞれのアクション名をパラメータとしてフロー制御部２０の実行要求を再帰的に行う（ステップＳ５８）。

【００３８】

ここで，不要なアクションの発火を避けるため，イベントオブジェクトに実行済みアクションリストを保存するメモリを設け，ここにリストされているアクション名が，選択されたアクションリストに含まれている場合には，それらのアクション名をアクションリストから削除し，１度実行されたアクションを実行しないようにしてもよい（ステップＳ５６）。もし，返送された実行結果に何のデータも含まれていなければ，フロー制御部２０は終了し，アクションの連鎖実行を停止する。

【００３９】

図９に，メッセージ送信部１６の処理フローを示す。メッセージ送信部１６では，何らかのメッセージ（送信要求）を受信すると，受信したメッセージの送信先アドレスをチェックし，自分宛かどうかを調べ（ステップＳ６０），その宛先が自分自身のアドレスであれば，自エージェントのメッセージ受信部１１に受信したメッセージを転送する（ステップＳ６１）。もしそうでなければ，メッセージの送信先アドレスに従って，他のエージェントにメッセージを転送する（ステップＳ６２）。

【００４０】

以下に具体的な実施例として，Java言語を用いて実現した場合の例を示す。なお，本発明はJava言語に特化したものではなく，他のプログラミング言語を用いて実装した場合についても適用可能である。

【００４１】

〔1〕フロー制御部２０（フローコントローラ）の実施例

フロー制御部２０はエージェントの振る舞いを既定するアクションの一つ（Fl

owController）のとして実装することができる。また，フロー制御部２０（以下，FlowControllerと記す）が発火する条件としてのメッセージパターンは，パラメータが一つであり，そのタイプがイベントオブジェクトであるとすると，その定義は以下のようになる。

【００４２】

```
public class FlowController extends Action {
    public void start(EventObject event) {
        //  フロー制御処理（図８参照）
    }
}
```

エージェントが，EventObject クラスのオブジェクトを受信した場合，上記FlowControllerのstart() メソッドが実行される。例えば，本実施例のstart() メソッドでは，引数のEventObject を以下のようなメッセージパターンに展開し，自分自身に送信する。

【００４３】

{ <イベントオブジェクト>，<イベントソース>，<イベント属性>，<直前のアクション名>，<直前のアクションの実行結果> }

ここで，イベントソースは，受信したイベントオブジェクトを最初に生成したエージェントのアドレスであり，イベント属性とは，イベントオブジェクトが持つ属性値であり，アクション連鎖の途中で変更されることを許容するものである。

【００４４】

《例１》アクション連鎖の例

以下のような２つのアクションがエージェントに追加されたとする。

【００４５】

```
public class ActionOne extends Action  {
    public Event_B  start(Event_A event) {
      :
    return new Event_B(...);
```

```
    }
  }
  public class ActionTwo extends Action  {
    public Event__C  start(Event__B event) {
              :
      return new Event__C(...);
    }
  }
```

ここで，Event__A，Event__B，Event__C は全てEventObject のサブクラスであるとする。すなわち，以下のようになる。

【００４６】

```
  public class Event__A extends EventObject  {... }
  public class Event__B extends EventObject  {... }
  public class Event__C extends EventObject  {... }
```

このとき，アクション管理部１２は，FlowControllerに合わせて，追加されるそれぞれのアクションの入力パターンを以下のように設定する。

【００４７】

ActionOne の入力パターン：

```
    { Event__A.class, AgentAddress.class, Object.class, String.class,
Object.class }
```

ActionTwo の入力パターン：

```
    { Event__B.class, AgentAddress.class, Object.class, String.class,
Object.class }
```

ここで，Event__A.class という表記は，クラス Event__A のオブジェクトなら何でもよいことを示す。同様に，String.classは文字列なら何でもよいことを示す。なお，FlowControllerの入力パターンは以下のようになる。

【００４８】

```
  {EventObject.class }
```

ここで，このエージェントに Event__A が送信された場合を考える。以下に説

明するように，アクションの連鎖は，結果的に図１０（Ａ）に示すようになる。このときのFlowControllerと，ActionOne およびActionTwo との関係を，図１０（Ｂ）に示す。

【００４９】

最初に受信されるメッセージの形式は，

{ event__A }

の形式であり， event__A はクラス Event__A のインスタンスであるとする。このとき，パターンマッチ処理部１３により，上記の３つの入力パターンの中でマッチするものとしてFlowControllerだけが選択され，実行される。FlowControllerは受信した event__A から以下のようなメッセージパターンを生成し，アクション実行要求を送信する。

【００５０】

{ event__A, event__A.source(), event__A.attribute(), null, null }

ここで，パターンマッチ処理部１３により，上記３つの入力パターンの中でマッチするものとしてActionOne だけが選択され，実行される。ActionOne はEvent __A を引数にして， Event__B を戻り値として返すため，FlowControllerは，次に以下のようなメッセージパターンを生成し，アクション実行要求を送信する。

【００５１】

{ event__B, event__B.source(), event__B.attribute(), "ActionOne", null}

ここで，パターンマッチ処理部１３により，上記３つの入力パターンの中でマッチするものとしてActionTwo だけが選択され，実行される。ActionTwo は Event__B を引数にして， Event__C を戻り値として返すため，FlowControllerは，次に以下のようなメッセージパターンを生成し，アクション実行要求を送信する。

【００５２】

{ event__C, event__C.source(), event__C.attribute(), "ActionTwo", null}

ここで，パターンマッチ処理部１３では，上記３つの入力パターンの中でマッチするものがないため，エラーメッセージをFlowControllerに返送する。よって，アクションの連鎖が停止する。

【００５３】

このようにして， Event__A の発生に対して，ActionOne →ActionTwo というデータフローを，それぞれのアクションをエージェントに追加するだけで定義することができる。この点が本発明の大きな特徴の一つである。

【００５４】

また，既存のデータフローに影響を与えずに新たなフローを追加したい場合にも，基本的にアクションを追加するだけでよい場合が多い。例えば，以下のような Event__A を入力とするActionTreeをエージェントにさらに追加した状況を考える。

【００５５】

```
public class ActionThree extends Action  {
    public Event__B  start(Event__A event) {
                :
        return new Event__B(...);
    }
}
```

ここで， Event__A が発生した場合，パターンマッチ処理部１３によって最初に選択されるアクションは，ActionOne とActionThree となるため，ActionOne →ActionTwo というフローとActionTreeというフローが同時に実行される。

【００５６】

アクションの連鎖は，結果的に図１１（Ａ）に示すようになる。このときのFlowControllerと，ActionOne ，ActionTwo およびActionTreeとの関係は，図１１（Ｂ）に示すとおりである。

【００５７】

《例２》アクション名によるフローの制約

上記の例１では，データフローを形成する全てのアクションが別々のイベント

種別を入力としていたため，問題はなかった。しかし，ActionTwo もActionOne と同じように Event__A の入力を期待するように定義されていると，全てのアクションが同時に発火してしまい，期待したデータフローが決定されない。この場合，それぞれのアクションを追加する際に，入力パターンをカスタマイズし，直前に実行されるアクション名を明示的に指定して，フローに制約を与えればよい。例えば，３つのアクションが全て，入力，出力とも Event__A を扱うものとする。

【００５８】

```
public class ActionOne extends Action  {
    public Event__A  start(Event__A event) {... }
}
public class ActionTwo extends Action  {
    public Event__A  start(Event__A event) {... }
}
public class ActionThree extends Action  {
    public Event__A  start(Event__A event) {... }
}
```

ここで，それぞれのアクションを追加する際，入力パターンの第４項目に具体的なアクション名を指定する変更を加える。

【００５９】

ActionOne の入力パターン：

{ Event__A.class, AgentAddress.class, Object.class, String.class, Object.class }

ActionTwo の入力パターン：

{ Event__A.class, AgentAddress.class, Object.class, "ActionOne", Object.class}

ActionThree の入力パターン：

{ Event__A.class, AgentAddress.class, Object.class, "ActionTwo", Object.class}

この状態で，エージェントに Event__A が発生したとする。入力パターンの第１項目は，いずれも Event__A.class であるため，全て候補になりうるが，FlowControllerが送信するメッセージの第４項目はnullであるため，最終的に選択されるアクションはActionOne だけになる。ActionOne の終了後，次にFlowControllerが送信するメッセージの第４項目は直前に実行されたアクションの名前“ActionOne”となるため，選択されるアクションはActionTwo だけとなる。

【００６０】

このようにして， Event__A に対して実際に決定されるデータフローは，ActionOne →ActionTwo →ActionThree となる。このアクションの連鎖の結果を図１２（Ａ）に示す。また，このときのFlowControllerと，ActionOne , ActionTwo およびActionTreeとの関係を，図１２（Ｂ）に示す。

【００６１】

《例３》入力パターンによるIF-THEN ルールの表現

パターンマッチの利点として，直列だけのフローを表現するだけでなく，一般的な条件分岐のフローを各アクションにまったく手を加えずに，入力パターンをカスタマイズするだけでよいというメリットがある。例えば，例２と同じ２つのアクションActionOne , ActionTwo に対して以下のようなBoolean タイプのオブジェクトを返すアクションを追加する場合を想定する。

【００６２】

```
public class ConditionalAction extends Action  {
    public Boolean start(Event__A event) {
if (...)
    return Boolean.TRUE;
else
    return Boolean.FALSE;
}
    }
```

それぞれの入力パターンは以下のとおりとし，ActionOne および，ActionTwo の入力パターンにおける第５項目の条件をそれぞれBoolean クラスのTRUEとFALS

E に変更している。

【００６３】

ActionOne の入力パターン：

{ Event__A.class, AgentAddress.class, Object.class, String.class, Boolean.TRUE }

ActionTwo の入力パターン：

{ Event__A.class, AgentAddress.class, Object.class, String.class, Boolean.FALSE}

ConditionalAction の入力パターン：

{ Event__A.class, AgentAddress.class, Object.class, String.class, Object.class }

この状態で， Event__A により最初に発火するアクションは，第５項目に条件がないConditionalAction だけであるが，このアクションが戻り値として，Boolean.TRUEを返すかBoolean.FALSE を返すかによって，次に実行されるアクションがActionOne になるかActionTwo になるかが決定される。すなわち，IF-THEN ルールのような条件分岐型のフローを入力パターンのカスタマイズにより実現可能となる。

【００６４】

本例におけるアクションの連鎖を図１３（Ａ）に示し，このときのFlowControllerと，ConditionalAction , ActionOne およびActionTwo との関係を，図１３（Ｂ）に示す。

【００６５】

〔2〕パーソナル・エージェントにおける適用例

本発明の具体的な適用例として，パーソナル・エージェントにおけるサービスイベントの動的通知の例を挙げる。ユーザは，様々なネットワークサービス（ｅ－ｍａｉｌ着信，電話着信，スケジュール更新など）のイベントをいつでも通知してもらいたい場合がある。また，ユーザは，場所や状況によって，パーソナルコンピュータ（ＰＣ）や携帯電話などの通信端末を頻繁に変更したい場合がある。パーソナル・エージェントの役目は，ユーザが現在使用中の通信端末を使って

，その端末に適切な表現方法で各種サービスのイベントをユーザに伝えることである。

【００６６】

例えば，ユーザがカレンダ・サービスからスケジュール更新イベントを受信した場合，「ユーザのＰＣがネットワーク接続されていればＰＣの画面上に表示し，そうでなければ携帯電話のショートメッセージでユーザに通知する」ようにしたい場合を考える。この場合のパーソナル・エージェント（ＰＡ）における，イベント通知のためのアクション構成を図１４に示す。

【００６７】

パーソナル・エージェント（ＰＡ）５１には，３つのアクション(IsPcOnline, SendToPHS, SendToPC) が追加されている。アクションIsPcOnlineはユーザが使用中のＰＣがネットワークに接続されている場合には，TRUEを返し，そうでなければFALSE を返すモジュールである。また，アクションSendToPCと，アクションSendToPHS は，それぞれ，ＰＣ端末とＰＨＳ端末に対する低レベル制御をエージェントでトラップしたソフトウェアであるＴＡ（ＰＣ）５２とＴＡ（ＰＨＳ）５３にイベントを転送するモジュールである。また，アクションSendToPCとアクションSendToPHS の入力パターンは，それぞれ以下のように設定されているものとする。

【００６８】

SendToPCの入力パターン：

{EventObject.class, AgentAddress.class, Object.class, String.class, Boolean.TRUE }

SendToPHS の入力パターン：

{EventObject.class, AgentAddress.class, Object.class, String.class, Boolean.FALSE}

この例は，前述の例３と同じルールで動作する。もし，ＰＡ５１がカレンダ・サービス (CalendarService)５４からスケジュール更新イベントを受信し，ユーザのＰＣがオンラインならば，ＰＡ５１内でIsPcOnline, SendToPCの順にアクションが発火し，スケジュールイベントはＰＣを制御するエージェントＴＡ（ＰＣ

）５２に転送される。エージェントＴＡ（ＰＣ）５２では，イベントが持つ情報をダイアログデータに変換するアクションDefaultScheduleToDialog が動作し，ダイアログデータを受信して画面に表示するアクションDisplayDialog が動作する。これにより，ＰＣの前に座っているユーザにスケジュール更新を知らせることができる。

【００６９】

また，ユーザのＰＣがオフラインなら，アクションIsPcOnlineはFALSE を返すため，アクションSendToPHS を経由して，イベントは携帯電話を制御するエージェントＴＡ（ＰＨＳ）５３に転送される。エージェントＴＡ（ＰＨＳ）５３では，イベントが持つ情報をショートメッセージに変換するアクションDefaultScheduleToSMSが動作し，変換されたショートメッセージデータを携帯電話網に送り出すアクションSMSendが動作する。これにより，携帯電話を持って外出しているユーザにスケジュールの更新を知らせることができる。

【００７０】

また，ＰＡ５１の３つのアクションにおける入力パターンの第１パラメータがEventObject.class で宣言されていれば，あらゆるイベントに対しての同一イベント通知ルールが適用されるため，スケジュール更新だけでなく，ｅ－ｍａｉｌ着信や電話着信などあらゆるサービスイベントに対してそのまま適用可能である。

【００７１】

逆に，特定のサービスについて，上記の通知ルールを適用したい場合には，そのサービスが送信してくるイベントのタイプに制限した入力パターンの一連のアクションをＰＡ５１に追加することにより，もともと定義したアクション連鎖に影響を与えることなくカスタマイズすることが可能である。

【００７２】

【発明の効果】

以上説明したように，本発明は以下のような効果を奏する。

【００７３】

（１）分散システムにおけるイベント処理において，イベントオブジェクト受

信時に実行されるアクションとサーバオブジェクトとを分離し，受信したイベントオブジェクトのタイプにより発火すべきアクションを選択するフロー制御手段をサーバオブジェクトに設けることにより，サーバオブジェクトにおける他のアクションに影響を与えることなく，動的に新たなイベント処理を導入することが可能となる。

【００７４】

（２）イベントオブジェクト受信時に実行されるアクションの定義とそのアクションが選択される条件としての入力パターンの定義を分離し，アクションの定義や構成を変えなくても，入力パターンの定義を変更することにより，イベントに対する振る舞いを変更することが可能となる。

【００７５】

（３）イベント受信時に実行されたアクションが，その実行結果としてイベントオブジェクトを返した場合，フロー制御手段が新たに受信したイベントオブジェクトのタイプをチェックし，次に発火すべきアクションの選択，実行を繰り返すことにより，動的なデータフローの決定が可能となる。

【００７６】

（４）入力パターンの定義にイベントオブジェクトのタイプだけでなく，イベントオブジェクトの値，またはイベントオブジェクトが持つ属性の値を設定することを許容することにより，アクションの発火を制御することが可能となる。

【００７７】

（５）入力パターンの定義に直前に実行されることが期待されるアクションの名前を設定することを許容し，フロー制御手段がアクションの選択時にこの定義をチェックすることにより，実行されるアクションの順序を制御することが可能となる。

【００７８】

（６）フロー制御手段がアクションの選択を行う際に，既に実行したアクションのリストを記憶する機能を設け，一度実行したアクションは，発火の対象にしないルールを設けることにより，データフローの無限ループを避けることが可能となる。

【００７９】

このように，分散システムにおけるオブジェクト間連携の仕組みとしてのイベント処理において，イベント処理定義部分をアクションという独立な部品として定義することにより，動的にその振る舞いを置き換えたり，既存の振る舞いに影響を与えずに新たな振る舞いを追加することが可能となる。さらに，フロー制御機能を加えることで，イベント処理定義を複数のアクションの連鎖で実現することを可能とし，これによって，イベント処理を構成するアクション部品の独立性を確保し，アクション部品の再利用性を高める効果がある。

【図面の簡単な説明】

【図１】

本発明のシステム構成例を示す図である。

【図２】

アクション・属性記憶部の構成例（その１）を示す図である。

【図３】

アクション・属性記憶部の構成例（その２）を示す図である。

【図４】

メッセージ受信部の処理フロー図である。

【図５】

アクション管理部の処理フロー図である。

【図６】

パターンマッチ処理部の処理フロー図である。

【図７】

アクション実行部の処理フロー図である。

【図８】

フロー制御部の処理フロー図である。

【図９】

メッセージ送信部の処理フロー図である。

【図１０】

アクション連鎖の例を示す図である。

【図１１】

アクション連鎖の例を示す図である。

【図１２】

アクション連鎖の例を示す図である。

【図１３】

アクション連鎖の例を示す図である。

【図１４】

パーソナル・エージェントにおけるイベント通知のためのアクション構成の構成例を示す図である。

【図１５】

従来のイベント処理モデルにもとづくクラス定義の例を示す図である。

【図１６】

従来のイベント処理シーケンスの例を示す図である。

【図１７】

アクション分離型イベント処理モデルにもとづくクラス定義の例を示す図である。

【図１８】

アクション分離型イベント処理シーケンスの例を示す図である。

【図１９】

アクション連鎖によるデータフローの例を示す図である。

【図２０】

従来技術によるアクション連鎖の実現例を示す図である。

【符号の説明】

１　動的データフロー決定装置（エージェント）

１１　メッセージ受信部

１２　アクション管理部

１３　パターンマッチ処理部

１４　アクション実行部

１５　アクション・属性記憶部

１６　メッセージ送信部

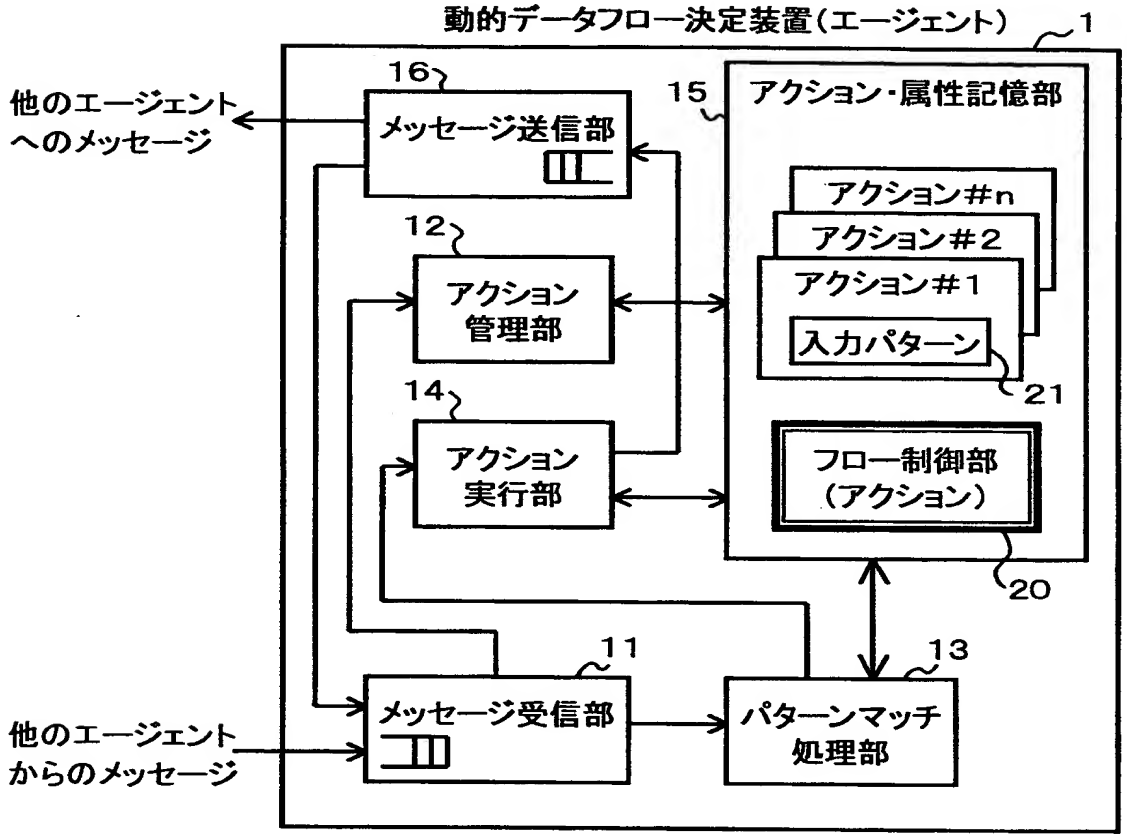
２０　フロー制御部

２１　入力パターン

【書類名】　図面

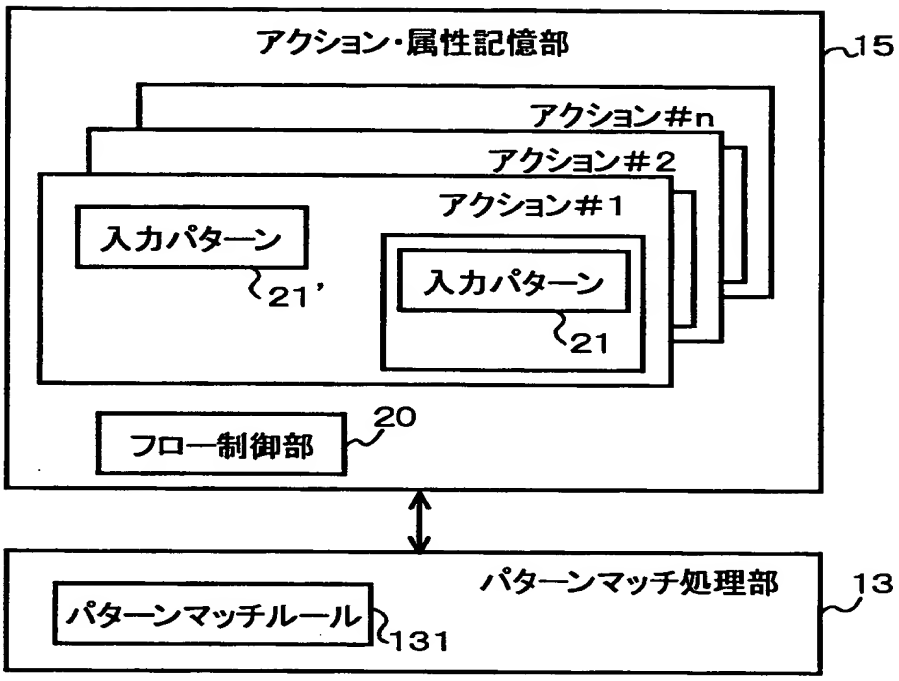
【図１】

本発明のシステム構成例



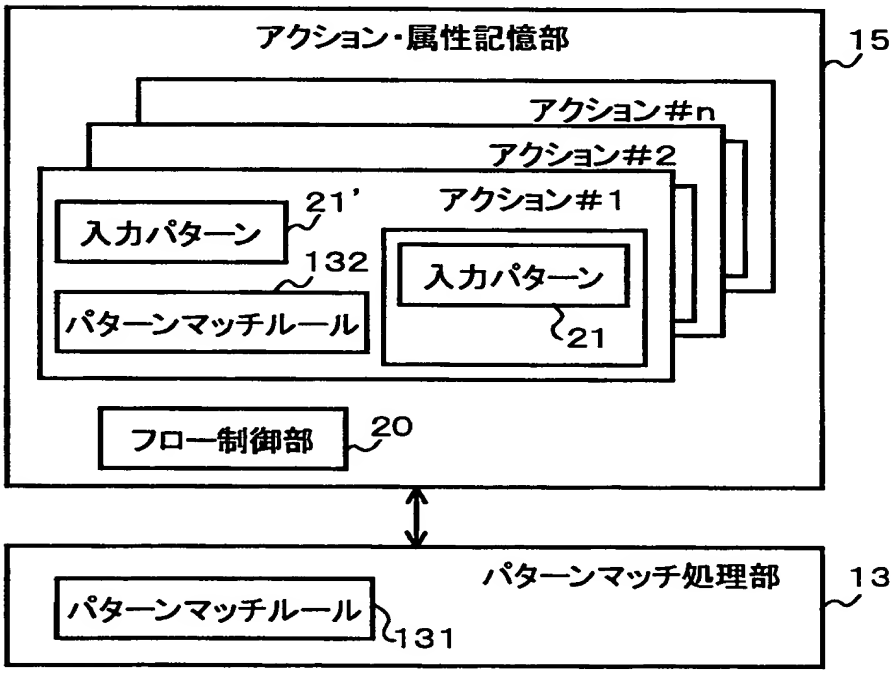
【図２】

アクション・属性記憶部の構成例（その１）



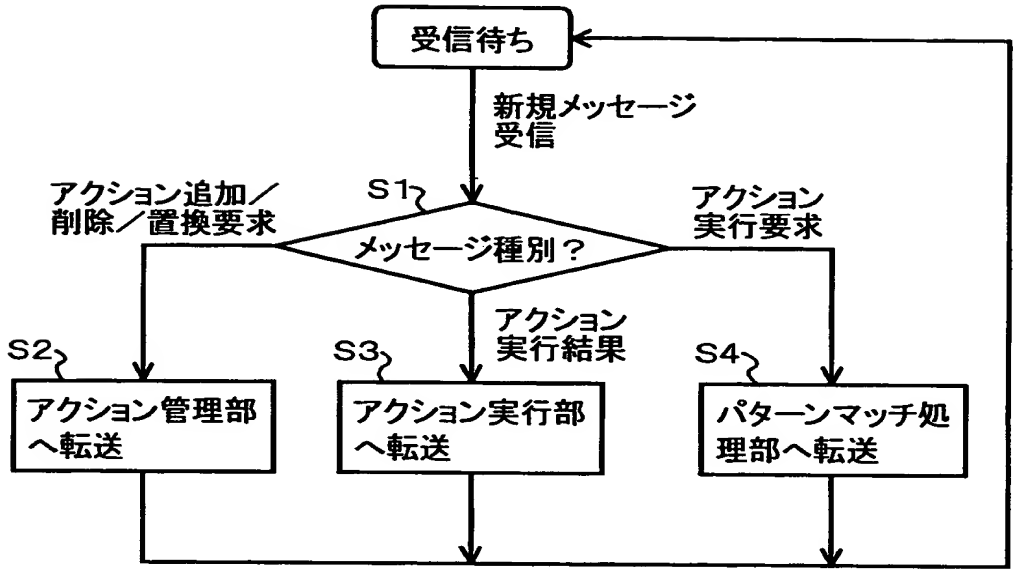
【図３】

アクション・属性記憶部の構成例（その2）



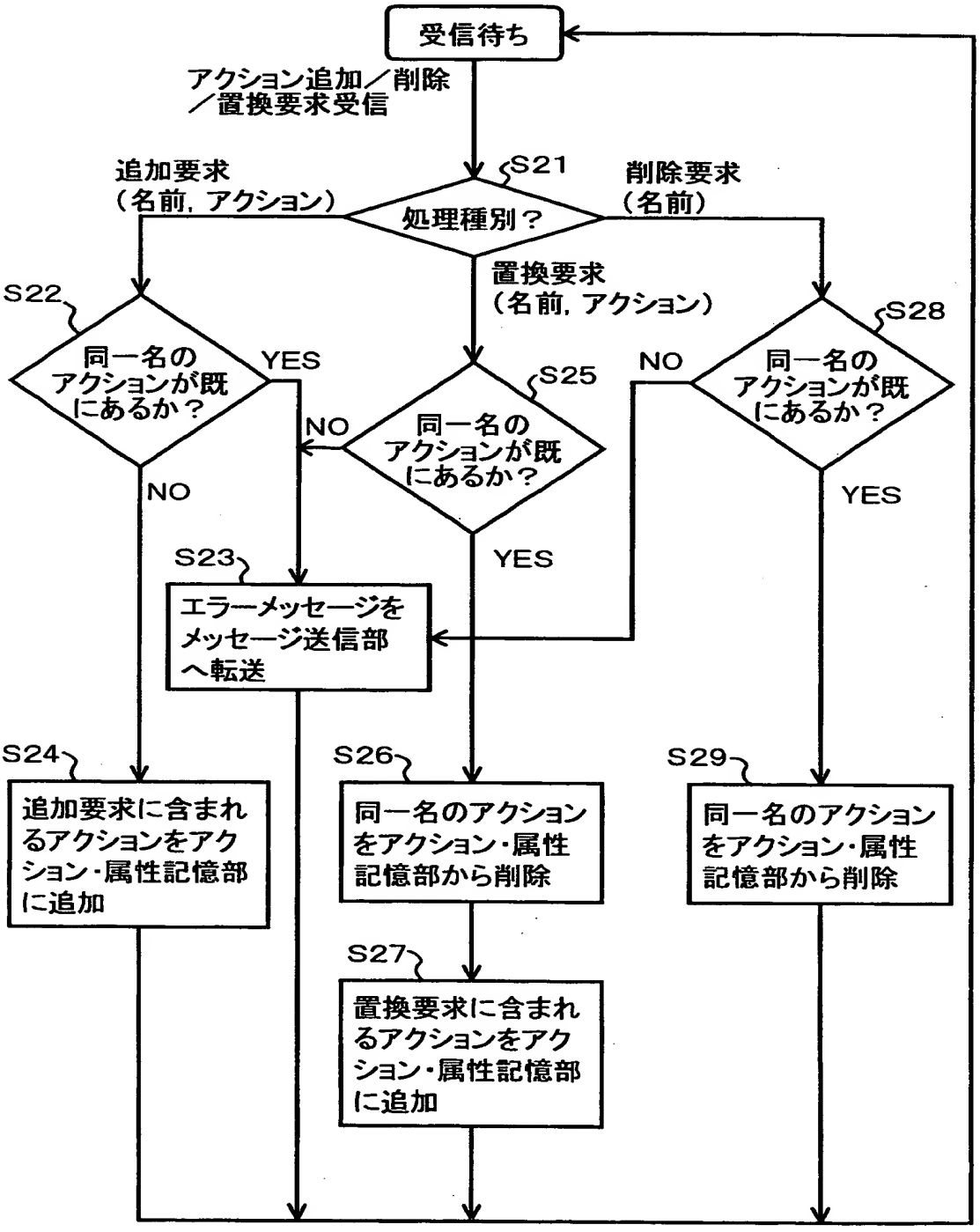
【図４】

メッセージ受信部の処理フロー



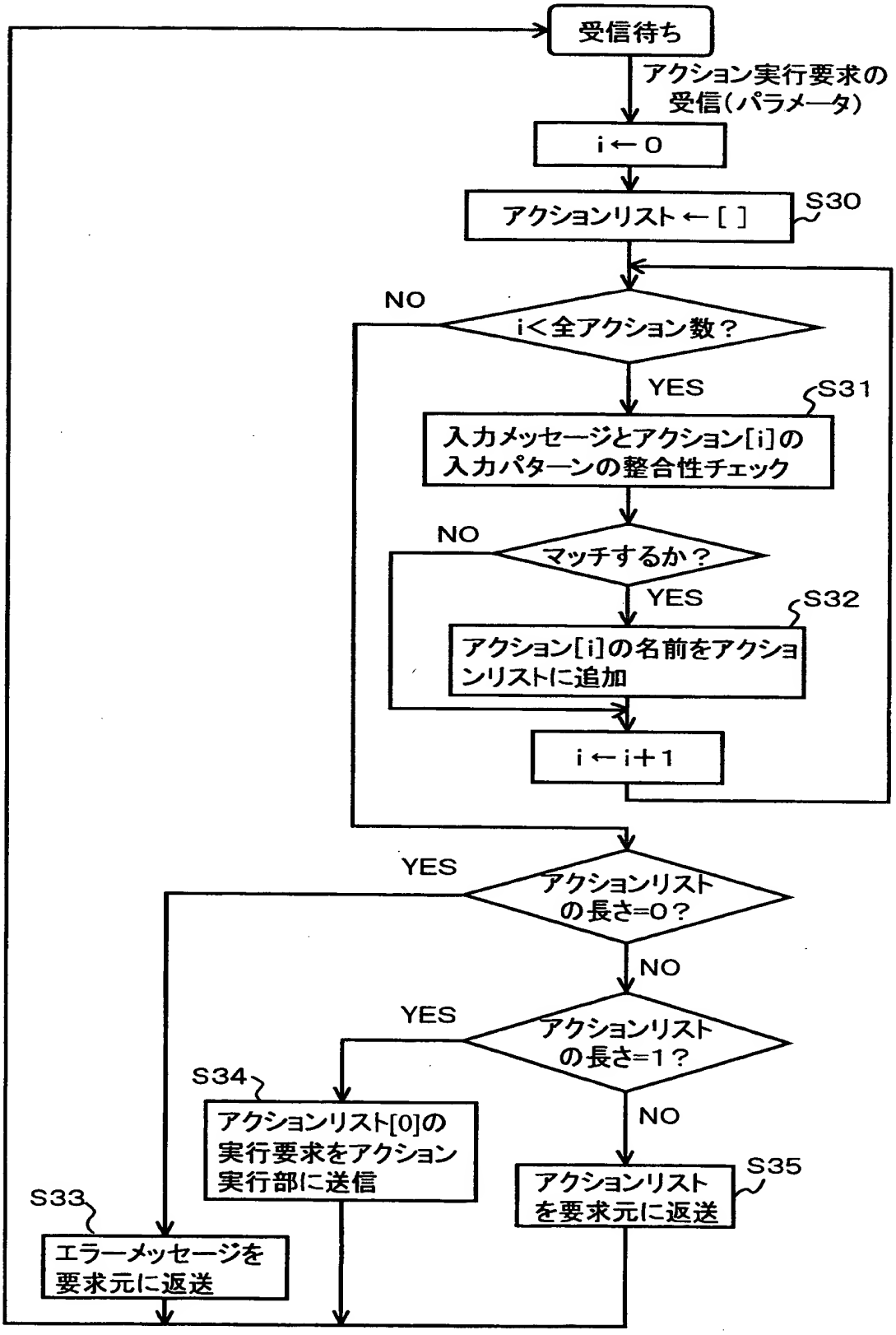
【図５】

アクション管理部の処理フロー



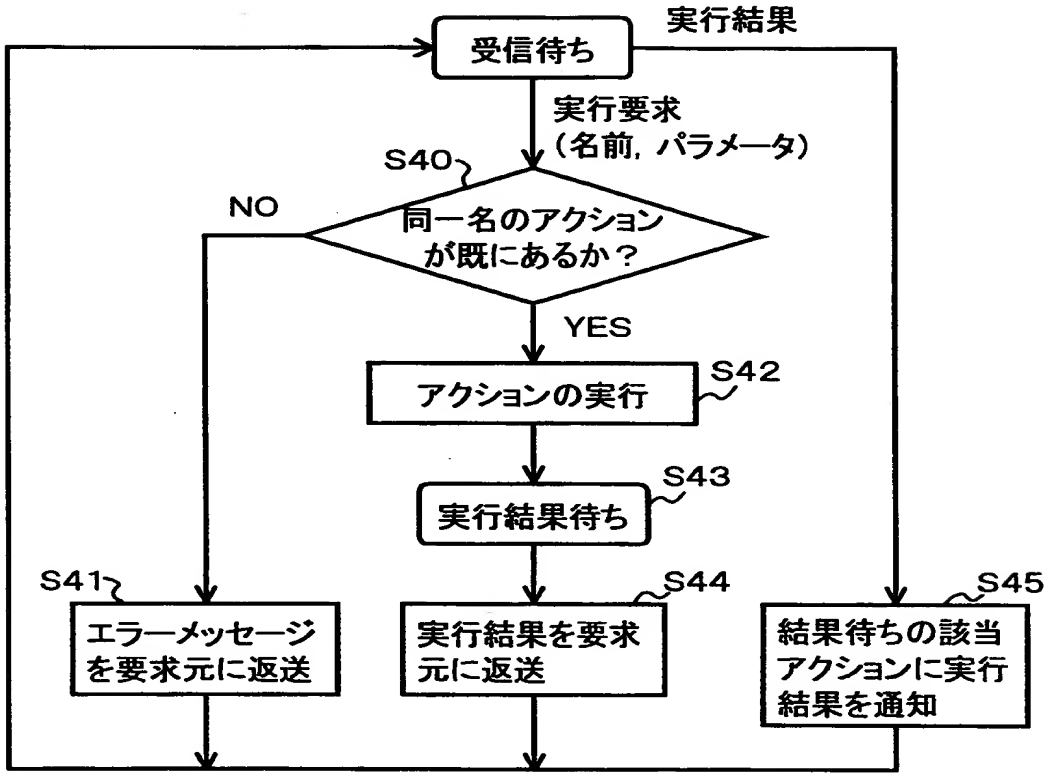
【図６】

パターンマッチ処理部の処理フロー



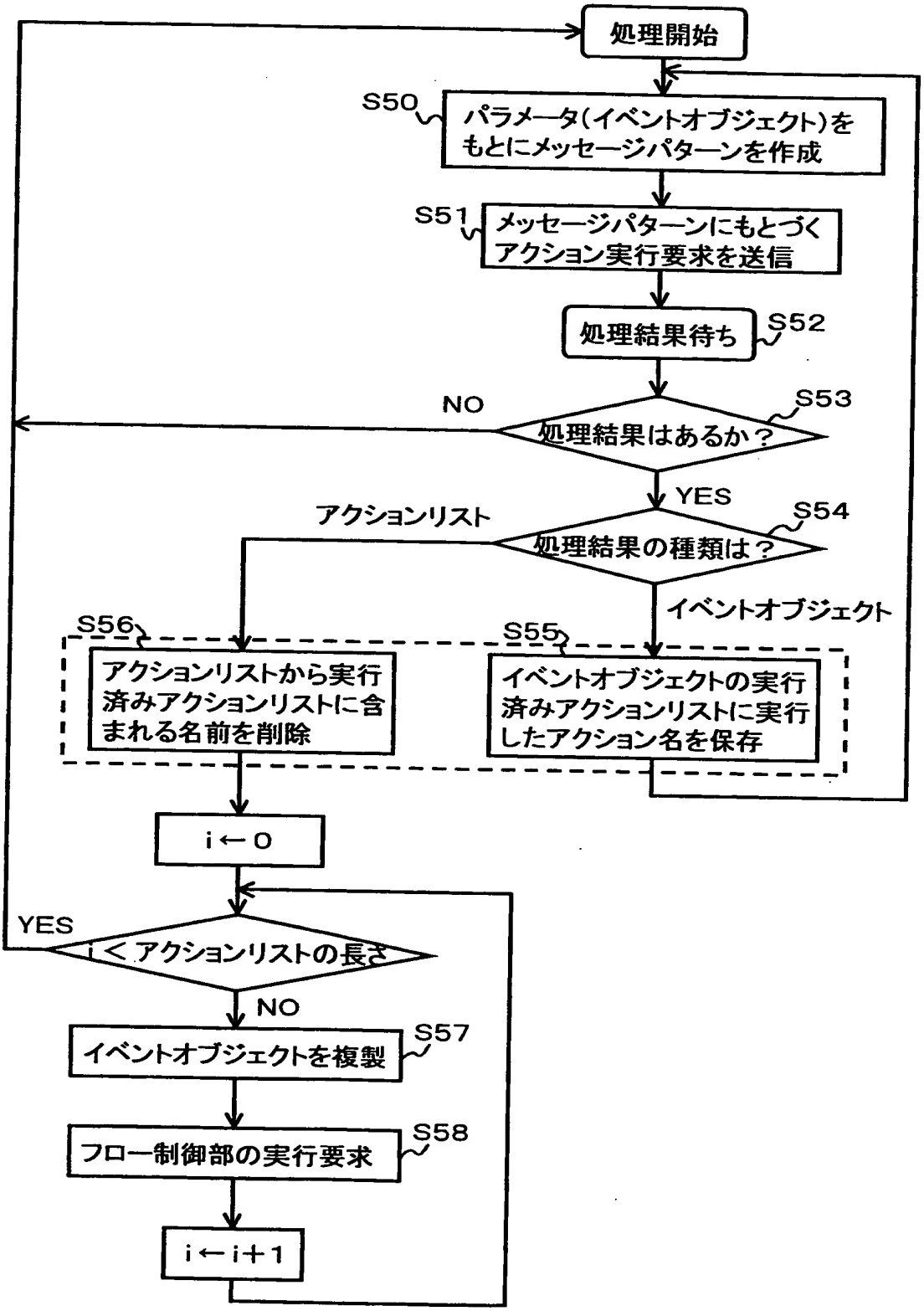
【図７】

アクション実行部の処理フロー



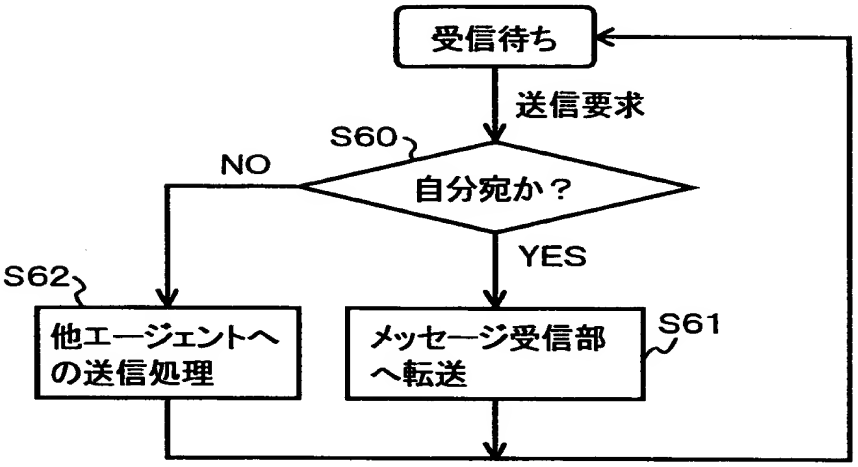
【図８】

フロー制御部の処理フロー



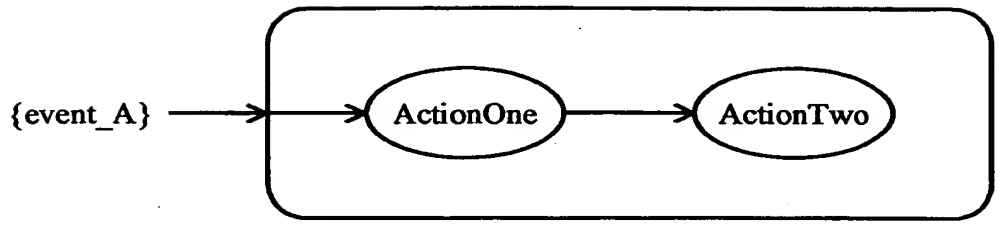
【図９】

メッセージ送信部の処理フロー

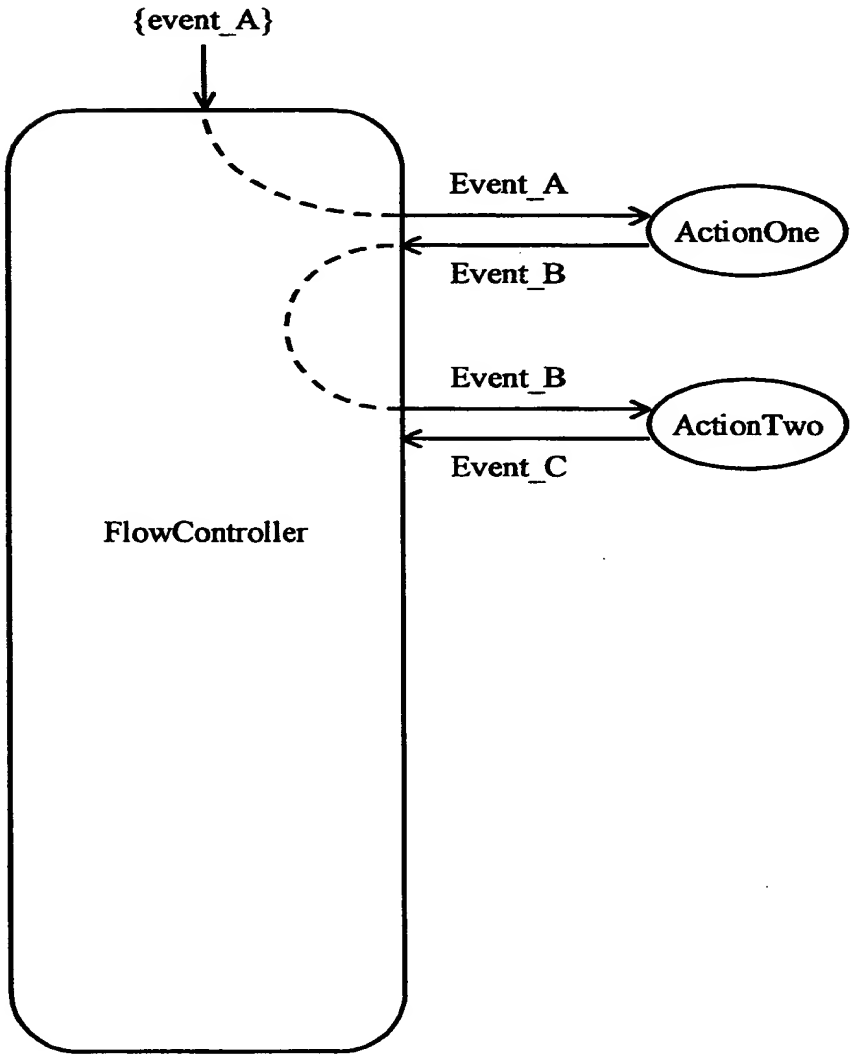


【図１０】

(A)

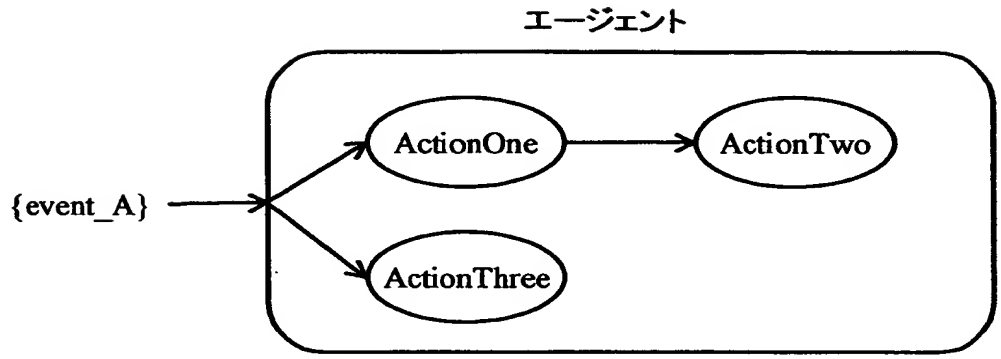

エージェント
{event_A}
ActionOne
ActionTwo

(B)

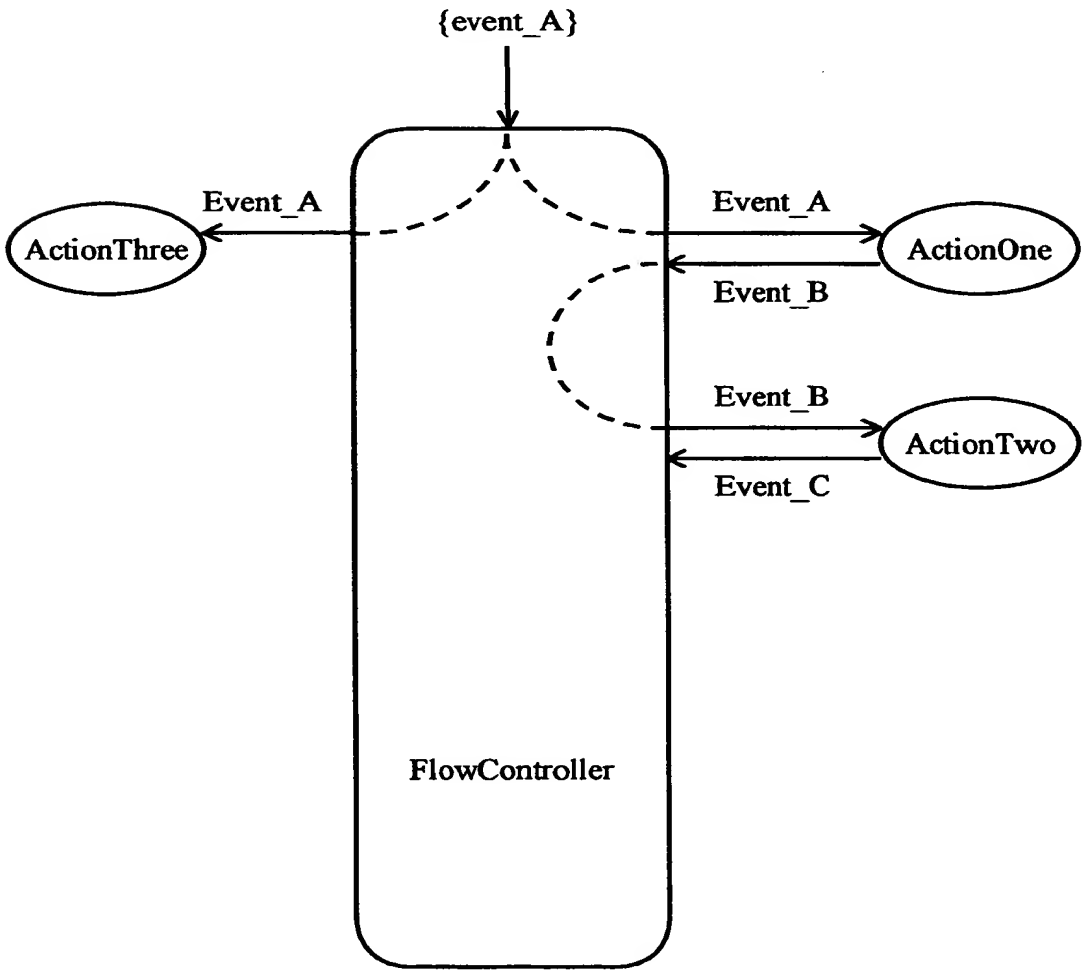

{event_A}
Event_A
ActionOne
Event_B
Event_B
ActionTwo
Event_C
FlowController

【図１１】

(A)

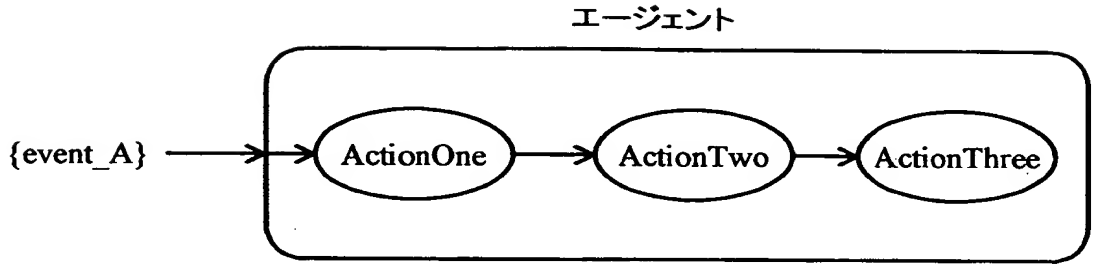

エージェント
ActionOne
ActionTwo
{event_A}
ActionThree

(B)

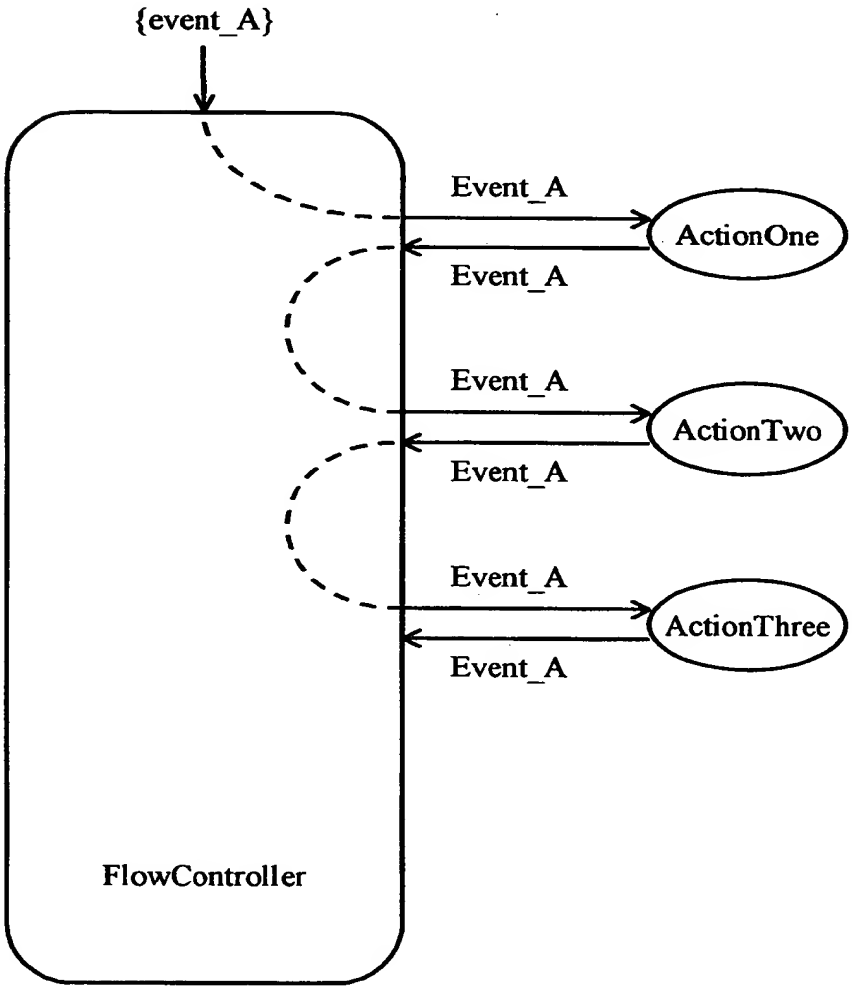

{event_A}
Event_A
ActionThree
Event_A
ActionOne
Event_B
Event_B
ActionTwo
Event_C
FlowController

【図１２】

(A)

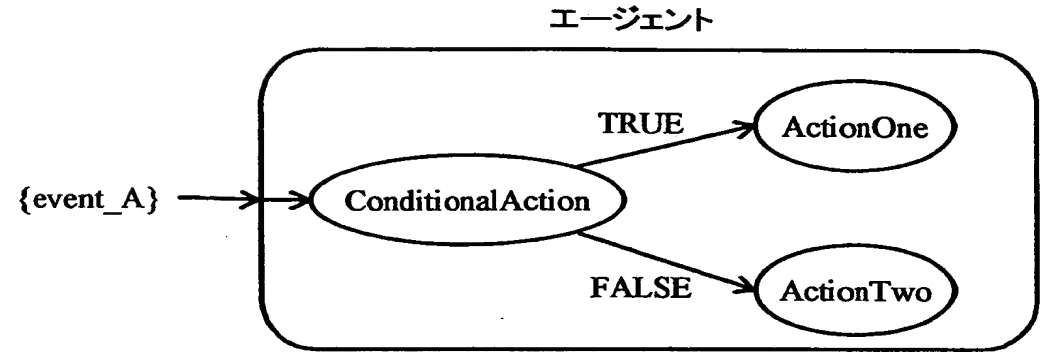

エージェント
{event_A}
ActionOne
ActionTwo
ActionThree

(B)

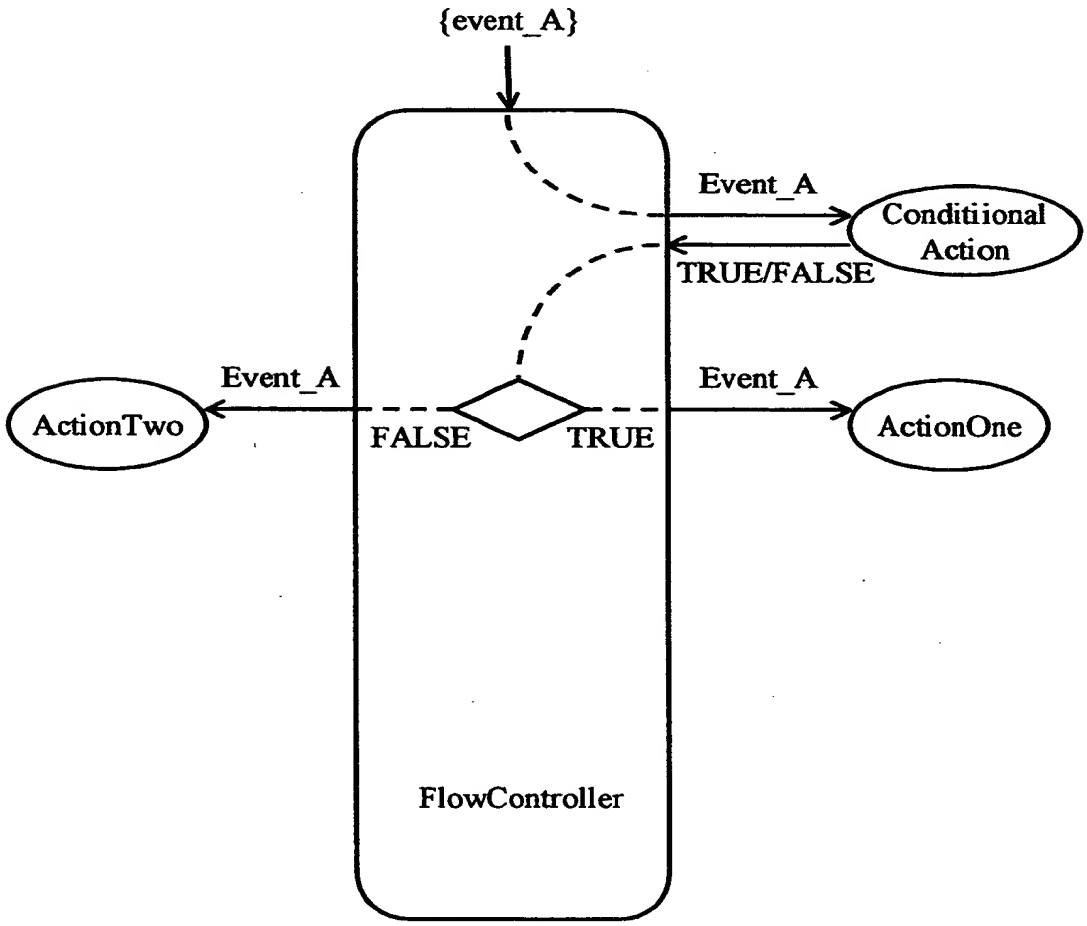

{event_A}
Event_A
ActionOne
Event_A
Event_A
ActionTwo
Event_A
Event_A
ActionThree
Event_A
FlowController

【図１３】

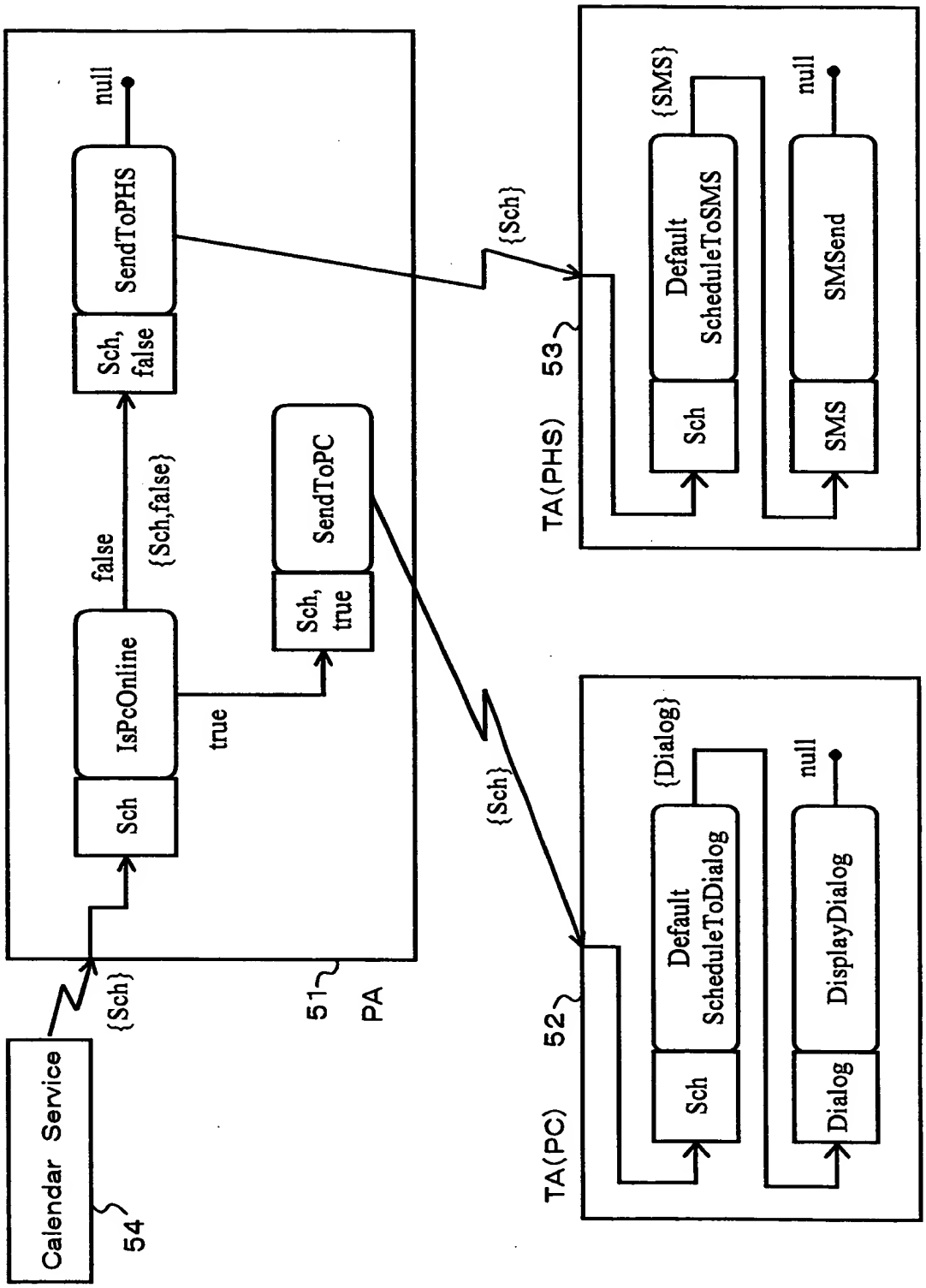
(A)


エージェント
TRUE
ActionOne
{event_A}
ConditionalAction
FALSE
ActionTwo

(B)


{event_A}
Event_A
Conditiional Action
TRUE/FALSE
Event_A
ActionTwo
FALSE
TRUE
Event_A
ActionOne
FlowController

【図１４】


Calendar Service
54
{Sch}
PA
51
Sch
IsPcOnline
false
{Sch,false}
true
Sch, false
SendToPHS
null
Sch, true
SendToPC
{Sch}
TA(PHS)
53
Sch
Default ScheduleToSMS
{SMS}
SMS
SMSend
null
{Sch}
TA(PC)
52
Sch
Default ScheduleToDialog
{Dialog}
Dialog
DisplayDialog
null

【図１５】

従来のイベント処理モデルにもとづくクラス定義の例

Source object

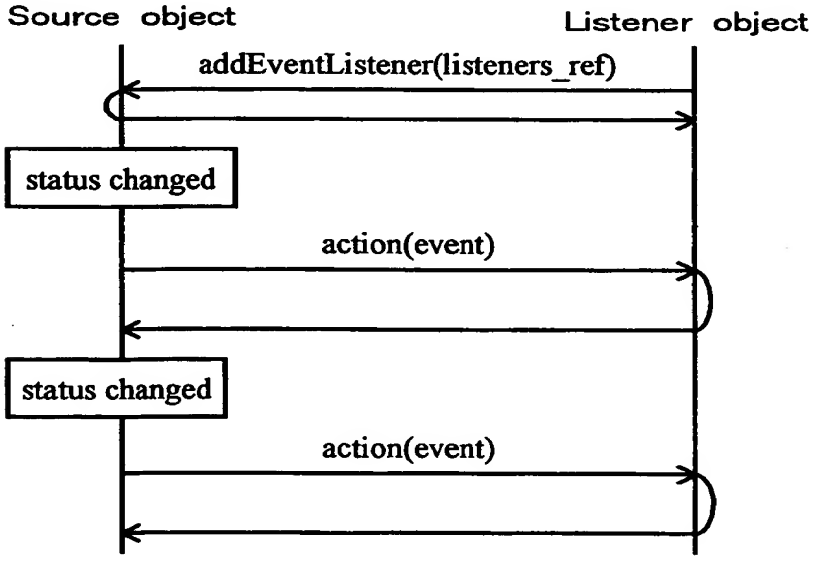
```
void addEventListner(Listener 1){
    listeners.add(1);
}

void removeEventListner(Listener 1){
    listeners.remove(1);
}

        ⋮

// Runtive phase
while (true) {
    if (status_changed){
        Event ev = ..... // prepare event
        for each listener in listener
            listener.action(ev);
    }
}
Vector listeners;
```

Listener object

```
void action(Event ev){
    ·····
}

        ⋮

// Initialization phase
Source s = ..... // obtain source refs.
s.addBarListeners(this);

        ⋮
```

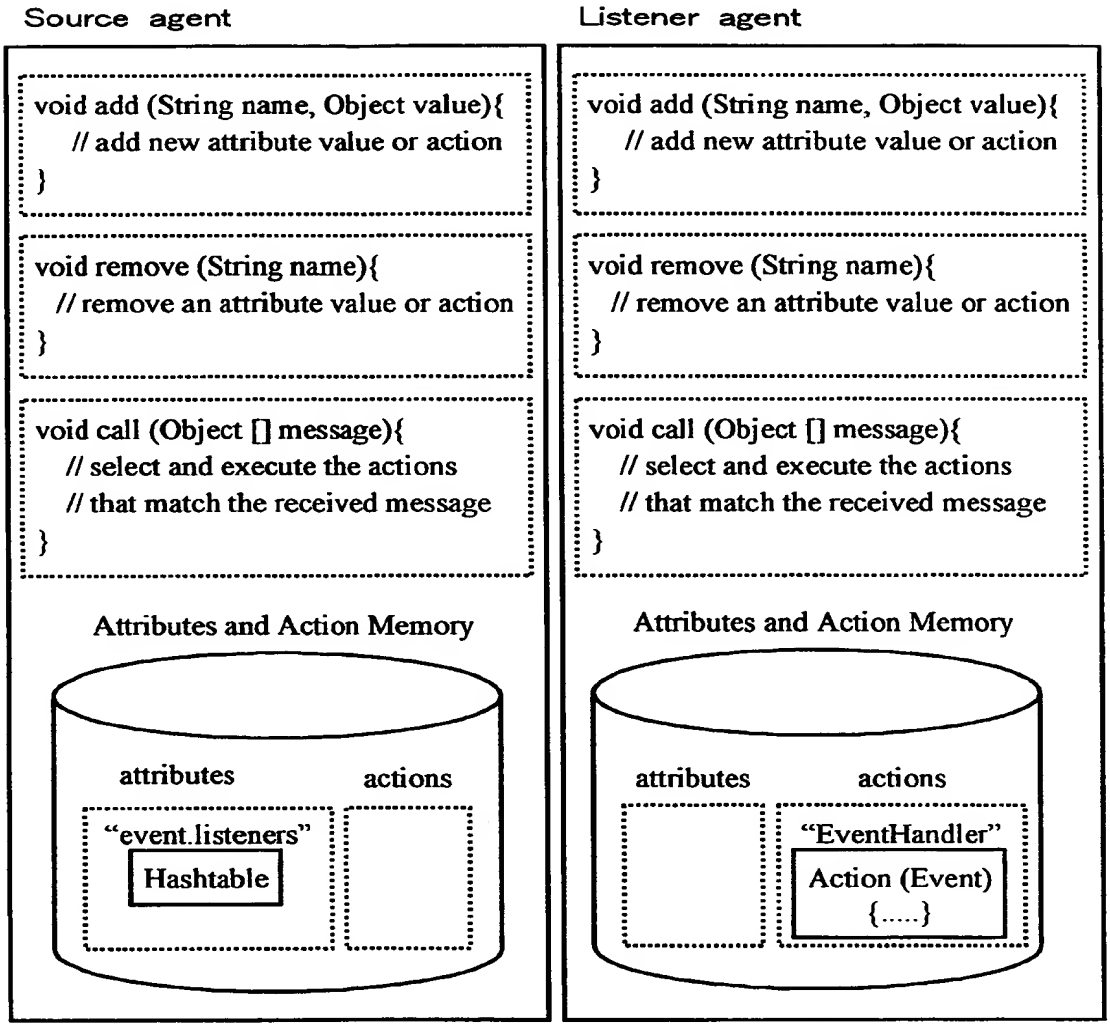
【図１６】

従来のイベント処理シーケンスの例



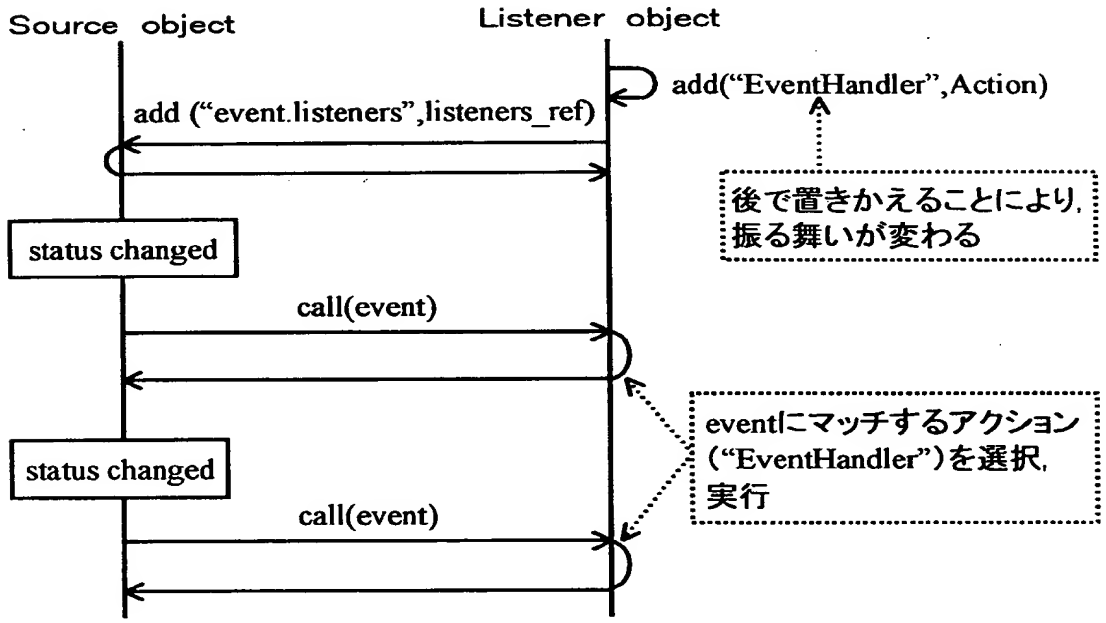
【図１７】

アクション分離型イベント処理モデルにもとづくクラス定義の例



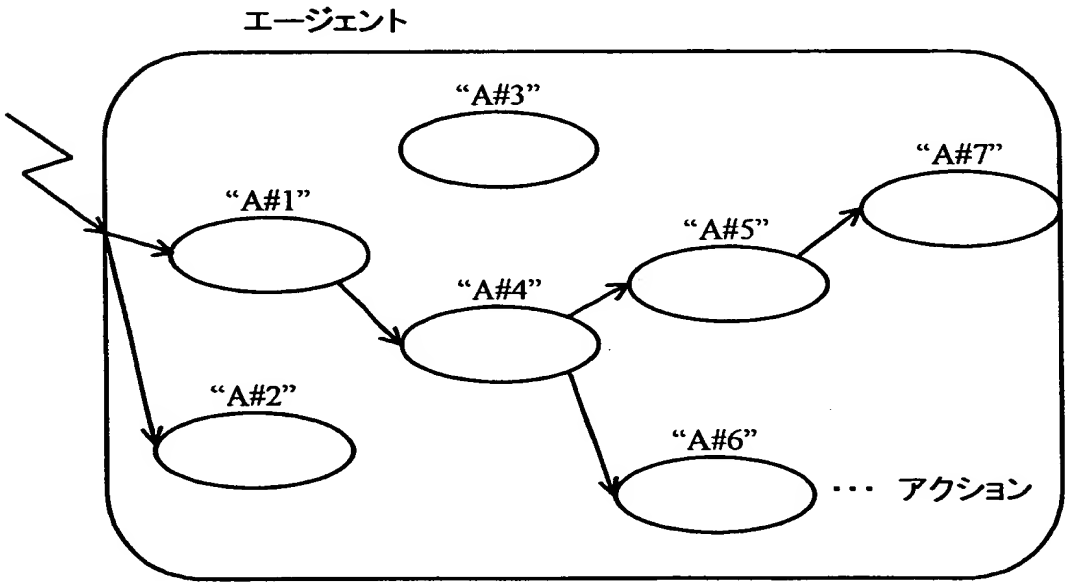
【図１８】

アクション分離型イベント処理シーケンスの例



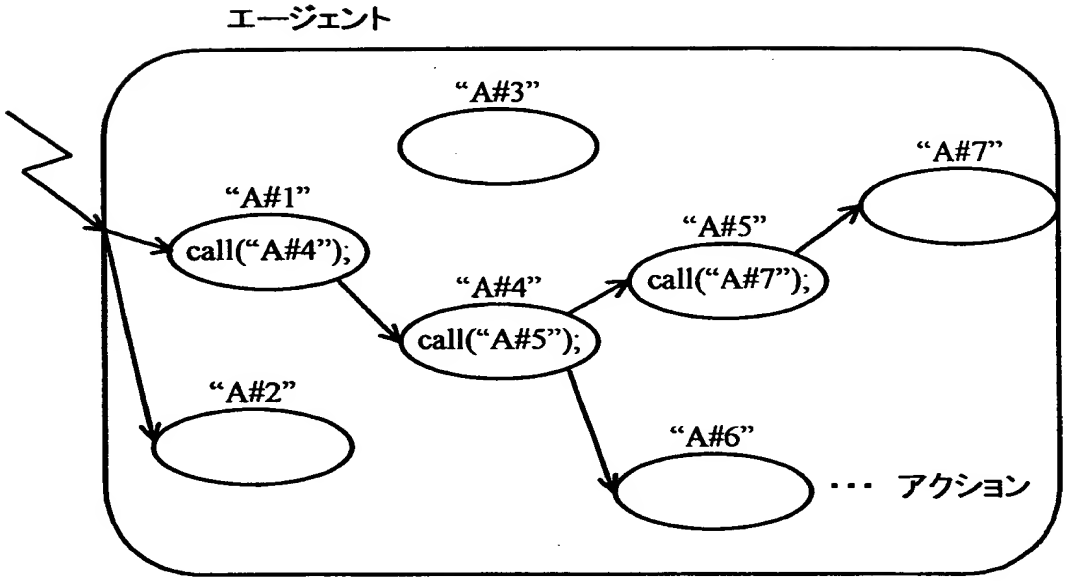
【図１９】

複数アクションの場合のアクション連鎖によるデータフローの例



【図２０】

従来技術によるアクション連鎖の実現例



【書類名】　　要約書

【要約】

【課題】　分散システムにおけるイベント処理におけるアクション連鎖による動的データフロー決定装置に関し，アクションの汎用性や独立性を確保しつつ，イベント処理を柔軟に構成・変更できるようにする。

【解決手段】　イベント処理時に実行される部分をアクションとして分離し，イベント処理定義をアクション部品から構成，変更可能とする。受信したメッセージがイベントオブジェクトであった場合，アクション実行部14は，フロー制御部20を発火する。フロー制御部20は，イベントの種類や状態に応じてエージェント内に存在するアクション部品の選択・実行を繰り返すことにより，アクション連鎖を生み出し，動的なデータフローを決定する。

【選択図】　　図１

出　願　人　履　歴　情　報

識別番号　　〔０００００５２２３〕

１．変更年月日　　１９９６年　３月２６日
［変更理由］　住所変更
住　所　神奈川県川崎市中原区上小田中４丁目１番１号
氏　名　富士通株式会社